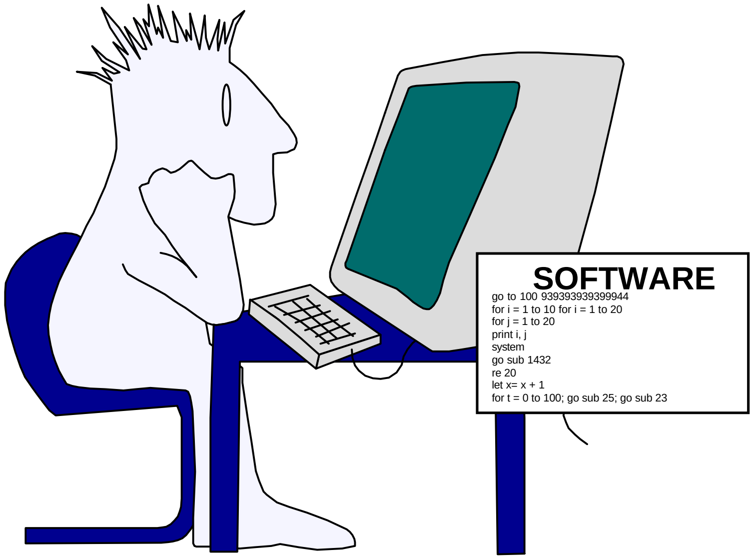
# SOFTWARE RELIABILITY & QUALITY




(14)

# SOFTWARE RELIABILITY

- **The probability that software will not cause the failure of a system or that software will not cause unanticipated conditions that could result in loss of system or subsystems.**

# WHY IMPORTANT?

- Tremendous growth in use of software (SW) to control systems.
- Software used to control critical life support and safety systems as well as entire unit (aircraft, nuclear power plant etc).
- Mechanical interlocks being replaced with SW
- Lack of discipline in generating SW now exists.
- Many critical accidents as a result of SW problems.
- Growth to continue.

# WHY IMPORTANT?

- General B. Randolph (June, 1989 AW&ST):
  - "... demand for software is growing at 25% per year."
  - "...cost and schedule growth are due to a failure of systems engineering and the requirements process..."
- Critical to weight savings in systems.
- Critical to elimination of personnel who could be used better elsewhere!

# SOFTWARE RELIABILITY OBJECTIVES

- WHAT ARE SOME SW RELIABILITY MODELS?
- WHAT TYPES OF PROBLEMS ARE THERE IN TRYING TO PREDICT SW RELIABILITY?
- WHAT “TYPES” OF SW EXIST (WHEN EVALUATING SW RELIABILITY AND SAFETY) AND WHAT ARE THE CRITERIA FOR EVALUATING THEM?
- WHAT TYPES OF ENVIRONMENTS DOES SW OPERATE IN?
- WHAT ARE SOME OF THE TYPES OF HARDWARE AND SW FAILURES?

# SOFTWARE RELIABILITY OBJECTIVES (con't)

- WHAT ARE SOME COMPUTER SYSTEM ERRORS THAT CAN OCCUR?
- WHAT ARE THE RISKS TO THE SYSTEM FROM SW?
- WHY DO ACCIDENTS INVOLVING SW HAPPEN--BOTH FROM THE SYSTEMS ENGINEER AND SW ENGINEERS VIEWPOINT?
- WHAT ARE SOME SW RELIABILITY or (SAFETY) AXIOMS THAT ARE NECESSARY TO PROPERLY UNDERSTAND SW?

# SOFTWARE QUALITY OBJECTIVES

- WHY DO ACCIDENTS FROM SW HAPPEN?
- WHAT ARE SOME SW QUALITY METRICS?
- WHAT TOOLS EXIST TO IMPROVE SW QUALITY?
- WHAT SHOULD SPECIFICATIONS FOR SW CONTAIN?
- HOW IS THE QUALITY AND RELIABILITY OF SW ASSESSED?

# SOFTWARE QUALITY OBJECTIVES(con't)

- WHAT WOULD YOU SPECIFY TO IMPROVE SW SAFETY?
- WHAT ARE THE TOOLS THAT AFFECT SW RELIABILITY AND HOW DO THEY AFFECT SW QUALITY?
- WHAT ARE FACTORS THAT AFFECT TRADEOFFS AND COSTING WHEN SW QUALITY IS EVALUATED?

# OUTLINE - SW RELIABILITY

## RELIABILITY

- Overview of the problem with SW.
- SW Reliability models.
- Types of SW.
- Sources of error.
- Tools to improve SW reliability & safety.
- SW safety axioms.

# OVERVIEW: WHERE DO FAILURES COME FROM?

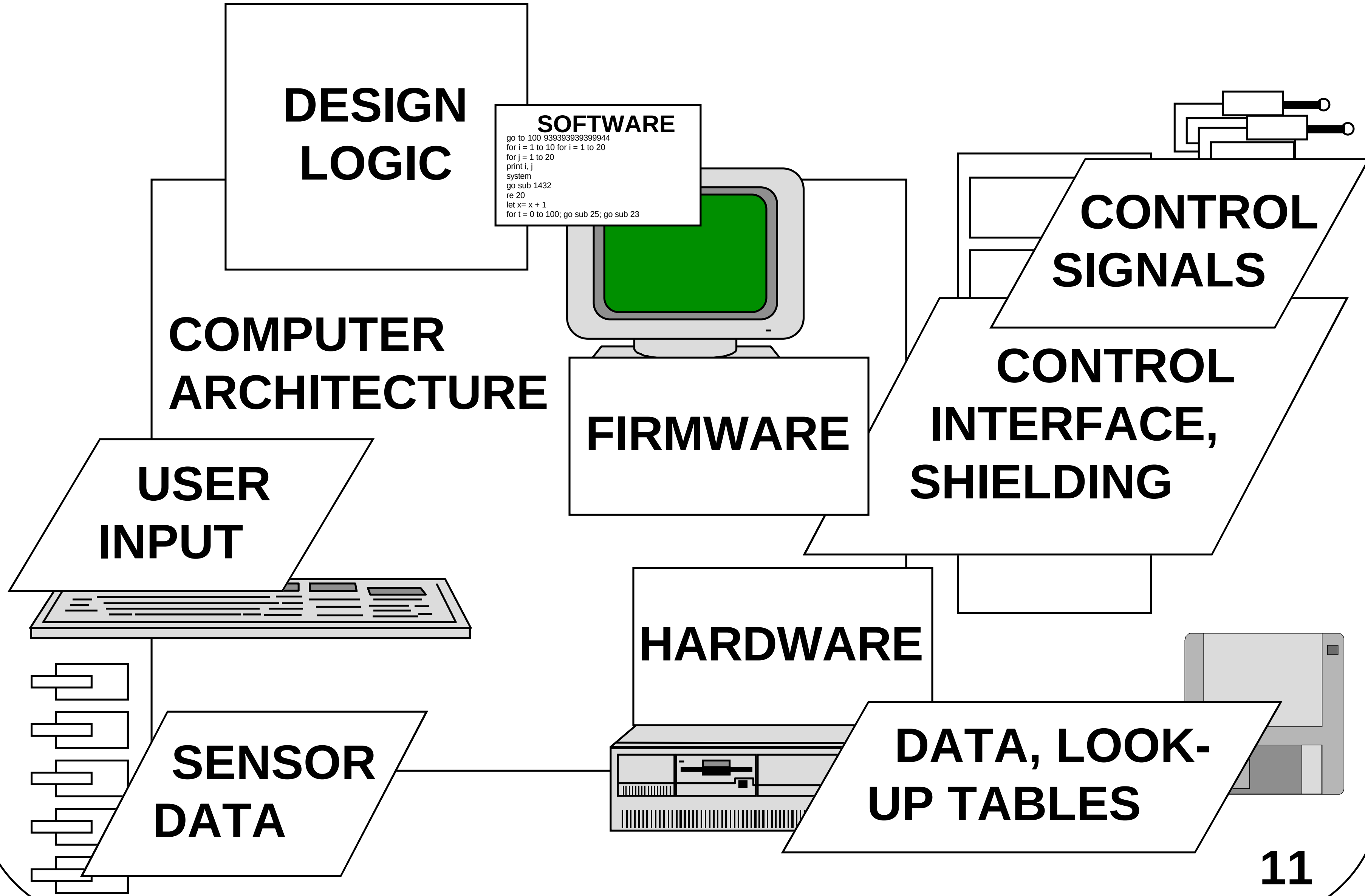
DESIGN

MANUFACTURE

**OPERATION**

***SOFTWARE***

# Problem Sources

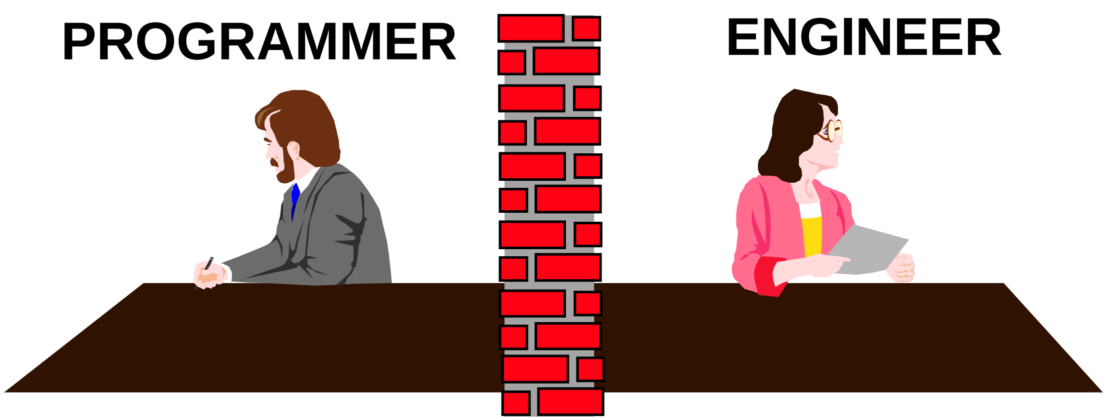


# OVERVIEW: HARDWARE /SOFTWARE RELIABILITY DICHOTOMY

- **There is a vast difference recognized between methods used for software, as opposed to hardware, to predict, inspect, test, assure, implement, and verify their reliability.**
- **This is due to the nonphysical abstract nature of software, the failures of which are always information design oversights or programming mistakes and are not based on environmental stresses or cumulative damage.**
- **As a discipline, software reliability uses few of the methods that apply to hardware reliability.**

# OVERVIEW: TRADITIONAL SOFTWARE ENGINEERING PROBLEMS

- LACK OF DISCIPLINE AND REPEATABILITY
- LACK OF DEVELOPMENT VISIBILITY
- CHANGING PERFORMANCE REQUIREMENTS
- LACK OF DESIGN AND VERIFICATION TOOLS
- LACK OF SOFTWARE REUSABILITY
- LACK OF COMMUNICATION



# SW RELIABILITY MODELS

- **Mathematical models which attempt to predict bug removal rate or number of bugs remaining based on testing or running time or bug count, etc.**
- **ASSUMPTIONS:**
  - **Perfect debugging.**
  - **All faults contribute to error rate.**
  - **Test Environment = Operating Environment.**
  - **Operating environment is static.**
  - **Errors are random and independent of past system behavior.**
  - **Availability of system improves as design errors are removed.**
  - **All tests can be conceived and anticipated.**

# SW RELIABILITY MODELS (Con't)

**Model categories are:**

- **Time domain: Software reliability is related to number of bugs at a given time during development.**
- **Data domain: Program reliability is estimated by running the program for a subset of input data.**
- **Axiomatic: It is postulated that reliability obeys certain universal laws.**
- **Other: Errors result from input data sets and logical paths etc.**

**P 14.2 (opt)**

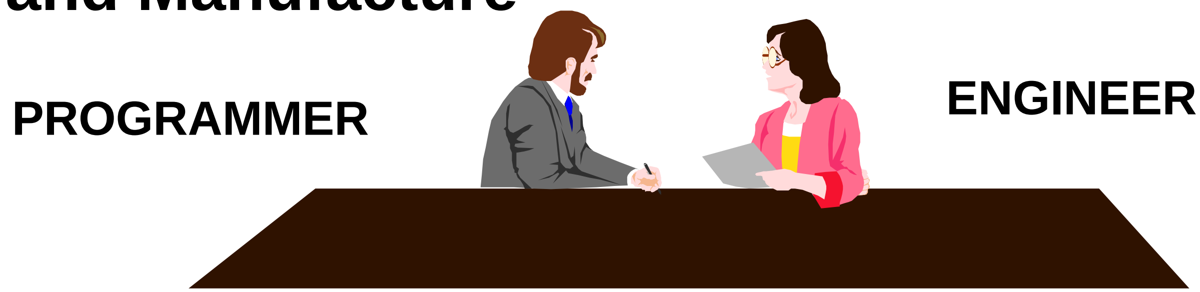
# Conclusions on Software Models

- Predicting reliability of SW and reliability modeling except for specialized highly structured cases is not an easy task.
- A quantifiable reliability approach is needed.
  - Clearly define software reliability and quality assurance functions.
  - Focus on complete elimination of critical defects and a specified tolerance level for minor defects.
  - Monitor and control defect removal and field performance.

# TOOLS TO IMPROVE SOFTWARE SYSTEM RELIABILITY & SAFETY

## ORGANIZATIONAL

- Communication/ Documentation/ Standardization/ Personnel/ Silver Bullets/ Configuration Management/ Software Reuse
- DESIGN & REQUIREMENTS
- Requirements/ Adding Features/ Anticipating Problems /Software-Hardware Interaction/ Isolating Processes
- OTHER PROBLEM AREAS
- Reliability/System/Sensor Interfaces/ RF Noise/ Maintenance and Manufacture



# SOFTWARE ANALYSIS TOOLS

- **Fault Tree Analysis (FTA).**
- **Petri Net Analysis.**
- **Hazards analysis.**
- **Formal logic analyzers.**
- **Software Failure Mode and Effect Analysis.**

# TYPES OF SOFTWARE
# Based on Timing & Control:

- **The allowability of REAL-TIME HUMAN ASSESSMENT and INTERFERENCE.**
- **Is the software AUTONOMOUS or INFORMATIONAL.**
- **Is the software TIME-CRITICAL or NON TIME-CRITICAL.**
- **For informational software is the info CRITICAL or NON -CRITICAL.**

# TYPES OF SOFTWARE
# Based on Timing & Control (con't)

## AUTONOMOUS, REAL-TIME CONTROL SW

- Real-time human evaluation of the program output or control activity is often not desirable. Real-time human interference is not desirable.

## AUTONOMOUS SOFTWARE

- Real-time human evaluation and response is possible and immediate corrective action may be necessary.

## INFORMATIONAL/ TIME CRITICAL SOFTWARE

- Real-time human actions and responses use the information to take immediate corrective action or to initiate other procedures.

# TYPES OF SOFTWARE
# Based on Timing & Control (con't)

## OPERATOR CONTROL SOFTWARE

- **Real-time human action is required for the program to control a system, initiate hazardous functions, or safety or protective activity, etc.**

## INFORMATION SOFTWARE

- **Human action and decisions are directly influenced by the information or the information is needed for safe operation of the system.**

# TYPES OF SOFTWARE
## Based on Environments

- Interactive
- Batch
- Remote job entry

# TYPES OF SOFTWARE
# Based on Categories

- Product software
- Embedded software
- Applications software
- Support software

# SOURCES OF ERRORS

- REQUIREMENTS/SPECIFICATIONS
  - Definitions / interpretations
- DESIGN
  - Hardware / software
- CODING
  - Source file entry / compilation
- INTEGRATION
  - Assembly / linking / file transfer
- MANUFACTURE
  - Duplication / loading / configuration

# SOURCES OF ERROR: "SOFTWARE ENGINEERING"

- LACK OF DISCIPLINE AND REPEATABILITY
- LACK OF DEVELOPMENT VISIBILITY
- CHANGING PERFORMANCE REQUIREMENTS
- LACK OF DESIGN AND VERIFICATION TOOLS
- LACK OF SOFTWARE REUSABILITY

# EXAMPLES OF COMPUTER SYSTEM ERRORS

- RADIATION MONITOR
  - TIming problem with data entry.
  - Hardware interlocks removed.
- CHEMICAL PLANT
  - Programmers did not understand process.
- SPACE SHUTTLE
  - Software revisions were not rechecked.
- AIRLINER
  - Personal computer shuts down navigation system.

# WHY DO ACCIDENTS HAPPEN?: SOFTWARE ENGINEER DEPENDENT

- **POOR AND INADEQUATE PRACTICE.**
- **NOT USING STATE OF THE ART TECHNIQUES.**
- **UNQUALIFIED PERSONNEL.**
- **DEPENDENCE UPON SILVER BULLETS.**
- **CHANGING REQUIREMENTS.**
- **SW ENGINEERS MAY NOT UNDERSTAND SYSTEM SAFETY.**
- **SYSTEMS MAY NOT BE MADE FAIL-SAFE OR HARDENED FROM EMI.**

# WHY DO ACCIDENTS HAPPEN?: SYSTEM SAFETY ENGINEER DEPENDENT

- **IGNORED SOFTWARE IN THEIR ANALYSIS OR LOOKED AT IT SUPERFICIALLY.**
- **EMPHASIZE DEPENDENCE ON NUMBERS.**
- **LITTLE INTERACTION WITH SW ENGINEERING PERSONNEL.**
- **OVEREMPHASIS ON FORM RATHER THAN CONTENT.**
- **OVERCONFIDENCE IN SOFTWARE.**

# TOOLS TO IMPROVE SOFTWARE SYSTEM RELIABILITY & SAFETY

- SOFTWARE MODULES/SOFTWARE REUSE.
- DISTRIBUTED REAL-TIME SYSTEMS.
- MULTIPLE VOTING SYSTEMS.
- MIX OF PROGRAMMING SKILLS & EXPERIENCE.
- ANALOG INTERLOCKS.
- ANALOG BACKUPS.
- SOFTWARE FAULT DETECTION.
- SOFTWARE ANALYSIS TOOLS.
- SOFTWARE DEVELOPMENT SPECIFICATIONS.

# TOOLS: SOFTWARE MODULES/SOFTWARE REUSE

- Reuse software
- Do not reinvent the wheel
- Keep senior programmers or senior managers who can review software.
- Modularized with well documented inputs and outputs..

# TOOLS: DISTRIBUTED REAL TIME SYSTEMS

- Multiple computers handle data analysis and I/O control capabilities.
- Developed as a fault tolerant system.
  - Error detection and correction.
  - Recovery procedures
  - Load balancing
  - Dynamic traffic time sharing

# TOOLS: MULTIPLE VOTING SYSTEMS

- Multiple systems are more reliable.
- Systems can sense when there are anomalies and can then alert operator.
- Systems need to have separate power and location (to avoid common mode failures).
- Systems for critical applications need to be separate from whistles and bells system for everything else.
- Keep SW to a minimum for critical control systems.

# TOOLS: MIX OF PROGRAMMING SKILLS & EXPERIENCE

- **Do not rely on all new programmers.**
- **Reuse software code when possible.**
- **Have software verification specialists.**

# TOOLS: ANALOG INTERLOCKS

- Do not replace mechanical or electro-mechanical interlocks with software interlocks (if at all possible).
- Keep analog backup systems for critical components.
- In querying alarms, assume there is a problem, set the alarms and then remove them one by one.

# TOOLS: ANALOG BACKUPS FOR CRITICAL SYSTEMS

- In special applications (e.g. a chemical process plant) allow actuators to go to a safe position if computer control is lost.
- Monitor the health of the backups and the output of software control commands independently of the main control computer.

# TOOLS: SOLUTIONS --FAULT DETECTION

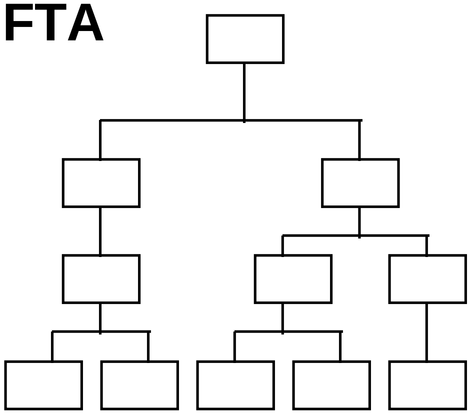
- PREVENTION OF HAZARDS THROUGH DESIGN.
  - Passive controls, designed to fail into safe state.
  - Reduce functionality if necessary.
- DETECTION AND TREATMENT AT RUN TIME.
  - Active Controls.
  - Designs which are conservative and assume errors to exist.

P 14.1 (opt)

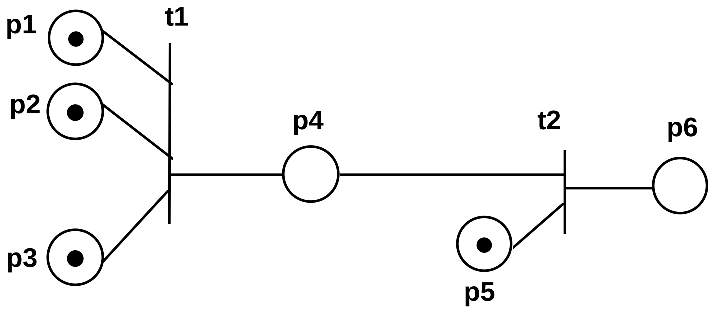
# SOFTWARE ANALYSIS TOOLS

- **Fault Tree Analysis (FTA).**
- **Petri Net Analysis.**
- **Software analysis programs.**
- **Other analysis tools.**

**FTA**



**Petri Net**



# TOOLS: SOFTWARE DEVELOPMENT SPECIFICATIONS

- WRITE A SOFTWARE SAFETY PROGRAM PLAN.
- WRITE A SOFTWARE SAFETY HANDBOOK AND RELIABILITY PRACTICES SPECIFICATIONS.
- DEVELOP A FORMAL PLAN FOR MAINTENANCE AND OPERATION.
- HAVE FORMAL SOFTWARE SYSTEM SAFETY WORKING GROUPS.
- INTEGRATE SYSTEM AND SW SAFETY, QUALITY AND RELIABILITY.

# SOFTWARE SAFETY AXIOMS

- TREAT SOFTWARE AS A SINGLE POINT FAILURE! (often in an analysis the software is just ignored).
- MANY ACCIDENTS ARE DUE TO INADEQUATE DESIGN FORESIGHT AND REQUIREMENTS SPECIFICATIONS:
  - Almost no accidents due to coding errors.
  - Incomplete or wrong assumptions about operation of controlled systems or requirements of operation of the computer.
  - Unhandled controlled system states and environmental conditions.

# MORE AXIOMS

- DECIDE WHAT YOU DON'T WANT TO HAPPEN -- MAKE SURE YOUR PROGRAM CAN'T GET THERE.
- MAKE YOUR SYSTEM FAULT TOLERANT.
- MAKE SURE YOUR SYSTEM CAN RECOVER FROM FAULTS.
- MAKE SURE YOUR SYSTEM CAN RECOVER FROM FAULTS.
- DO NOT KEEP CHANGING THE SYSTEM SPECIFICATIONS.

# MORE AXIOMS

- IT IS IMPOSSIBLE TO BUILD A COMPLEX SW SYSTEM TO BEHAVE EXACTLY AS IT SHOULD UNDER ALL CONDITIONS!
- SOFTWARE SAFETY, QUALITY & RELIABILITY ARE DESIGNED IN, NOT TESTED IN.
- COMPLACENCY MAY BE THE MOST IMPORTANT ROOT CAUSE OF ACCIDENTS.
- UPSTREAM APPROACHES TO SOFTWARE SAFETY ARE MOST EFFECTIVE.
- SW ALONE IS NEITHER SAFE OR UNSAFE.
- MANY SW BUGS ARE TIMING PROBLEMS WHICH ARE DIFFICULT TO TEST FOR.

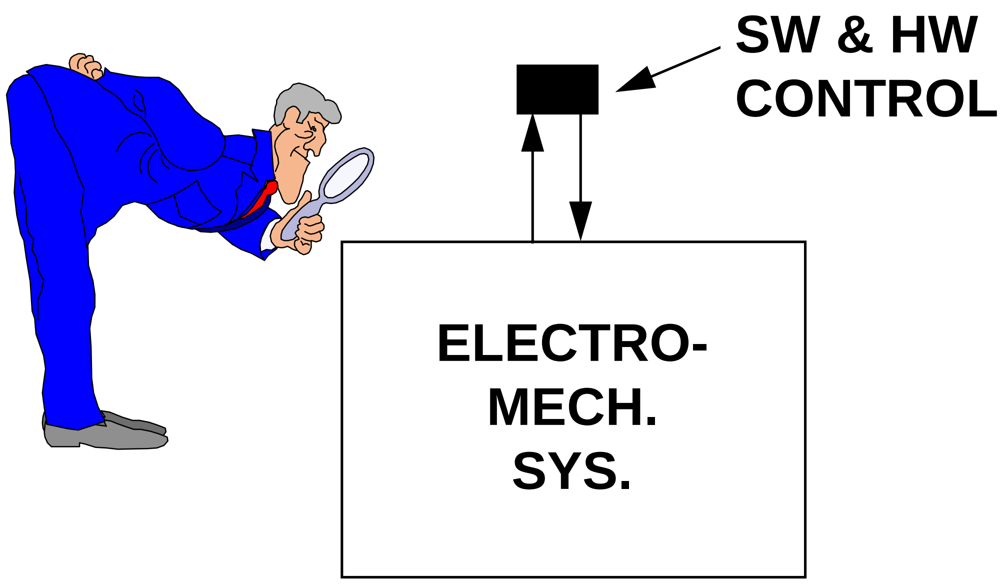
# MORE SUGGESTIONS & AXIOMS:

- Keep safety critical systems as small as possible and simple as possible by moving any functions that are not safety critical to other computers.
- As a rule software systems do not work well until they have been used.
- Software is the weak link.
- Mathematical functions implemented by SW are not continuous functions but have an arbitrary number of discontinuities.
- In many organizations there is lack of up-to-date professional standards in software engineering (and/or lack of use).

# MORE SUGGESTIONS & AXIOMS

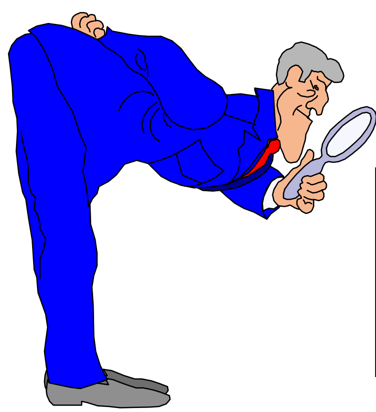
- **Engineers believe one can design black box tests without knowledge of what is inside the box.**
- **Use independent IV&V**
- **Use IBM clean room approach to develop SW.**
- **Software often fails because the SW goes somewhere that the programmer does not think it can get to!**

# CONCLUSION: Not a Black Box



# CONCLUSION: Single Point - Unstable Failure Source.

**Buffer overload, timing issues, sneak circuits, discontinuities in algorithms, whistles and bells, more whistles and bells, system overload, HW failure, sensor failure, rf interference, unstable & undocumented programs, multiple languages, programmers do not understand engineering system, sabotage, memoryless batch programs, HW errors, HW voltage level anomalies, clever software, overly complex software, no IV&V, no revision testing, poor bug reporting, no modularity**



**ELECTRO-MECH. SYS.**

**SW & HW CONTROL**

# CONCLUSIONS

- Software is used in critical applications.
- Software is a weak link in the system reliability chain.
- The potential problems with software are not well understood.
- If handled properly and applied properly the application of software/ hardware can be a valuable design option.
- There are many ways to validate and improve software (see also software quality).

END